Conway’s Game Of Life

By Pradosa Patnaik

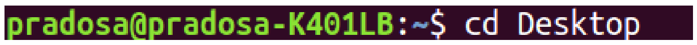
Instructions

The can be run from a bash terminal

1.Download the GameOfLife.zip file

2.Navigate to the directory where the zip file has been downloaded

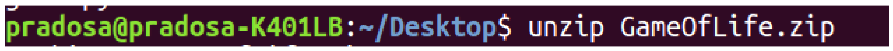
Command : cd YourDirectoryName



```
pradosa@pradosa-K401LB:~$ cd Desktop
```

3.Unzip the zip file with the below command

Command: Unzip GameOfLife.zip



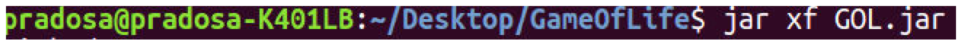
```
pradosa@pradosa-K401LB:~/Desktop$ unzip GameOfLife.zip
```

4. Navigate to the GameOfLife directory

Command: cd GameOfLife

5. Extract the GOL.jar with the below command

Command: jar xf GOL.jar



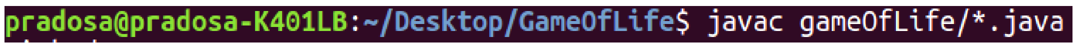
```
pradosa@pradosa-K401LB:~/Desktop/GameOfLife$ jar xf GOL.jar
```

6. Navigate to the GOL directory

Command: cd GOL

7. Compile the Java files with the below command

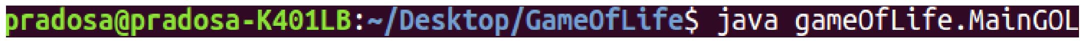
Command: javac gameOfLife/*.java



```
pradosa@pradosa-K401LB:~/Desktop/GameOfLife$ javac gameOfLife/*.java
```

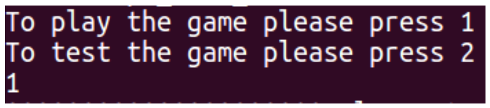
8. Run and test the game with the below command

Command: java gameOfLife.MainGOL



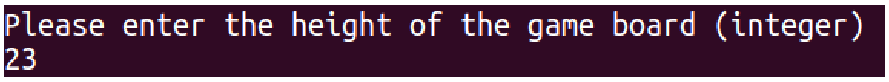
```
pradosa@pradosa-K401LB:~/Desktop/GameOfLife$ java gameOfLife.MainGOL
```

9. To Play the Game press 1 and hit Enter



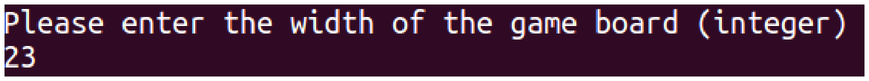
```
To play the game please press 1
To test the game please press 2
1
```

10. Enter the height of the game board and press enter



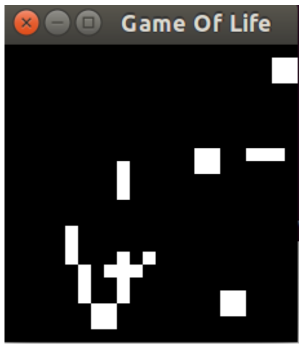
```
Please enter the height of the game board (integer)
23
```

11. Enter the width of the game board and press enter



```
Please enter the width of the game board (integer)
23
```

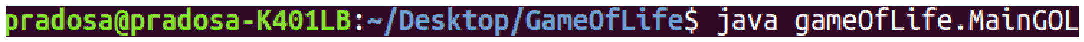
12. Now the game screen will appear on the screen displaying the evolving states of the cell after each iteration.



13. To stop the game press the close button on the top left corner of the game window.

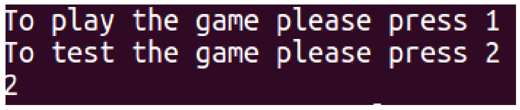
Test the correctness of the game implementation :

14. To test the correctness of the game implementation run the game again with the below command.



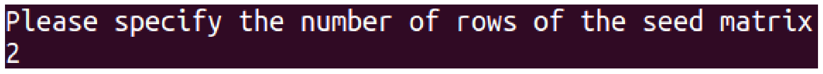
```
pradosa@pradosa-K401LB:~/Desktop/GameOfLife$ java gameOfLife.MainGOL
```

15. Press 2 and hit enter



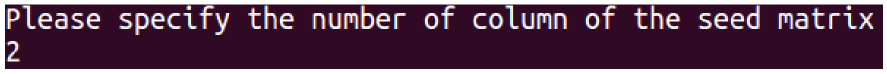
```
To play the game please press 1
To test the game please press 2
2
```

16. Enter the number of rows of the seed matrix (Integer value)



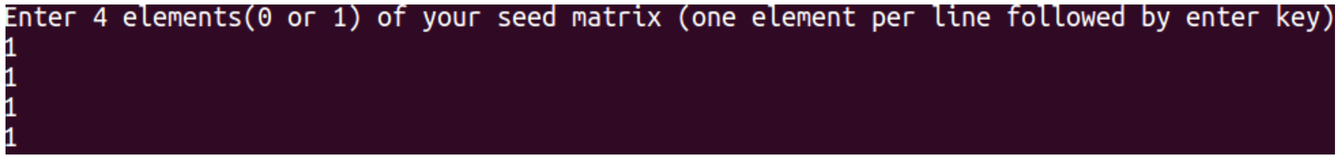
```
Please specify the number of rows of the seed matrix
2
```

17. Enter the number of columns of the seed matrix (Integer value)



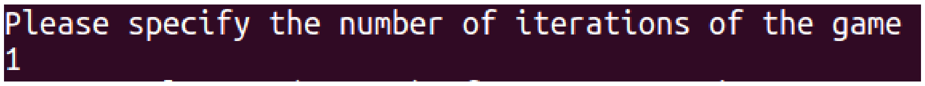
```
Please specify the number of column of the seed matrix
2
```

18. Enter the elements of the seed matrix (0 or 1) one value at a time followed by enter key

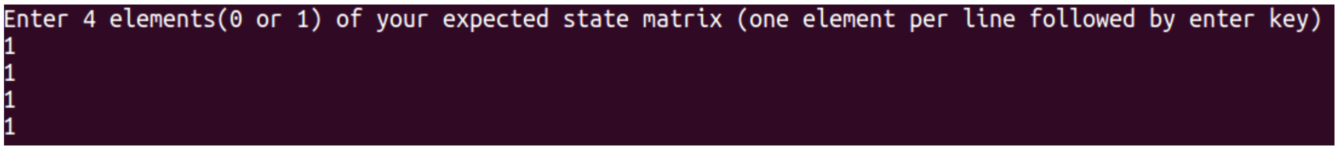


```
Enter 4 elements(0 or 1) of your seed matrix (one element per line followed by enter key)
1
1
1
1
```

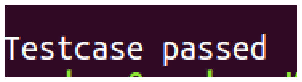
19. Enter the number of iterations of the game


```
Please specify the number of iterations of the game
1
```

20. Enter the elements of the expected_state matrix (0 or 1) one value at a time followed by enter key


```
Enter 4 elements(0 or 1) of your expected state matrix (one element per line followed by enter key)
1
1
1
1
```

21. The testcase result will be shown on the terminal as follows


```
Testcase passed
```

Discussion

I used the following technologies: Java, Java awt libraries to design the game board and display the progress of the game.

Design

The design of the game is has been kept simple. Below is the class description of the classes and interfaces used in the project.
There are total three classes in the project.
1. MainGOL
2. GameBoard
3. GameTest

MainGOL:

- This class serves as the entry point to start the game. This class contains the main method to take input from command line, create the object of GameBoard and start the game.
- The main method also has the option to test the game in test-mode.

Member Functions :

- ■ **Main Method (public static void main(String[] args):**
- main method takes input from standard input , creates the object of GameBoard and start the game.
- The main method also has the option to test the game in test-mode.

GameBoard:

- This class is the main class to initialize the GameBoard and has necessary methods to run and display the progress of the game on screen.

Data Members:

int frameSize1 : Height of the Jframe window

int frameSize2 : Width of the Jframe window
BufferedImage image : Image to show the sates of the game
int[] pixels : Array to hold the reference to the image pixels
Random randomGen : Random number generator to initialize the game board
double generationSpeed : This parameter specifies the speed at which the next state of the game are generated
int height : specify the height of 2D array
int width : Specify the width of 2D array
int[][] current : 2D array to hold the current state of the game
int[][] next : 2D array to hold the next state of the game

Member Functions :

- **GameBoard(int height, int width):**
  Constructor to initialize the current state of the GameOfLife grid with a given height(height) width(width)
- **GameBoard(int [][] seed):**
  Constructor to initialize the Current State of the GameOfLife from a given seed (2D array of 0s and 1s)
- **int[][] NextGeneration():**
  Method to generate the next state of the game from the current sate of the game.
- **int decideNextState(int x, int y):**
  Method to decide next state of the cell from the current cell value and the 8 surrounding cell values as per the given criteria
- **void start():**
  Method to start the game to display on the screen
- **void render()**
  Method to render the next state of the cell value to the image pixels
- **void initBoard()**
  Method to initialize the game board using a image of size height(height) and width(width)
- **void dispaly()**
  Method to display the evolution of cells in the board on the screen as the the game advances

GameTest:

- This class contains necessary method to test the game

Data Members:

GameBoard golTest : Height of the Jframe window

Member Functions:

- **boolean test_game(int[][] seed, int num_of_iteration,int[][] expected_state)**
  Method to test the implementation of the game.

Requirements

Build a simulation of Conway's game of Life, which is a infinite two dimensional grid of square cells which can take binary states either 0 (dead) or 1(alive) . The Game follows the simple rules to generate the next generation of the game.

Every cell interacts with its eight neighbors which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

- Any live cell with fewer than two live neighbours dies, as if caused by under-population.

- Any live cell with two or three live neighbours lives on to the next generation.
- Any live cell with more than three live neighbours dies, as if by over-population.
- Any dead cell with exactly three live neighbours becomes a live cell, as if by
- reproduction.

Assumptions:

- The game assumes an infinite grid. To make things simple, we are working with a finite two dimensional array (N*M) array.
- We provided a method to to display the evolving animation of the game on screen.
- The initial state of the game is randomly initialized with 0s and 1s with the help of a random value generator.
- The cells on the edges do not have 8 neighbors as our grid is not infinite. This case can be handled in many ways but for simplicity we are considering the actual neighbors of the array (less than 8 neighbors) and the next state of the these cells are decided based on the current state of the existing neighbors.
  Example: The next state of the cell located at coordinate (0,0) is decided based on the state of two of its neighbors located at (0,1) and (1,0) .

Testing

To test the implementation we have created several unit test cases and made sure the unit test cases cover the basic requirement of the game. The test plan and the unit test results have been provided in a separate file. Also a test method with signature test_game (seed,num_of_iteration,expected_state) is provided to test the implementation.